CS240A Project: Parallelized Attacks on Elliptic Curve Discrete Logarithm Problem

Qin ZHOU
fatestudio@gmail.com

Liang XIA
liangxia2006@gmail.com

March 20, 2013

1 Abstract

The infeasibility of Elliptic Curve Discrete Logarithm Problem (ECDLP) guarantees the security of all Elliptic Curve based cryptography. There are many attacking methods towards ECDLP to degrade this infeasible problem, among them Pollard ρ method is the best general method known so far[1]. In this project, we generate many elliptic curves according to our demand, then develop two algorithms: naive bruteforce algorithm and Pollard ρ algorithm, and parallelized both algorithms. After that we tested all algorithms and evaluated the power of parallelization.

2 Introduction

Elliptic Curve Cryptography (ECC) is a new public-key technology that offers performance advantages at higher security levels. ECC can be applied into many public-key protocols, e.g., an Elliptic Curve version of the Diffie Hellman key exchange protocol (ECDH)[2], Elliptic Curve Digital Signature Algorithm (ECDSA)[3] and Elliptic Curve versions of the ElGamal Signature Algorithm [4]. These protocol prototypes build the foundation of current computer security.

ECC uses much less bits while achieving the same level of security compared to RSA (160 bits of ECC provide the same security as 1024 bits of RSA), which means less bandwidth usage and better performance. Figure 1 gives the bit number of ECC and RSA at the same level of security.

The adoption of ECC has been slower than had been anticipated, perhaps due to the lack of freely available normative documents and uncertainty over intellectual property rights[5].

3 Elliptic Curve Cryptography

An elliptic curve E is the graph of an equation

$$E : y^2 = x^3 + ax^2 + bx + c$$

where a, b, c are in whatever is the appropriate set (rational numbers, real numbers, integers mod a prime, etc).

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

Figure 1: Bit number of ECC and RSA at same level of security

There are many different families of elliptic curves, among which we define our elliptic curves below:

$$E = \{(x, y)|x, y \in \mathbb{F}_p, y^2 = x^3 + ax + b\}$$

Where x, y, a, b are in finite field $\mathbb{F}_p$, p is a prime number. We choose a and b randomly and they satisfy the equation below to make the curve non-singular:

$$4a^3 + 27b^2 \equiv 0 \ (mod\ p)$$

After the definition of elliptic curve E, we define the addition operation on points of E:

Suppose two points $P_1 = (x_1, y_1) \in E$ and $P_2 = (x_2, y_2) \in E$, define addition operation:

$$P_3 = (x_3, y_3) = P_1 + P_2 = (x_1, y_1) + (x_2, y_2)$$

where

$$x_3 = m^2 - x_1 - x_2$$
$$y_3 = m(x_1 - x_3) - y_1$$

and

$$m = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & P_1 \neq P_2 \\ (3x_1^2 + b)/(2y_1) & P_1 = P_2 \end{cases}$$

If the slope m is infinite, then $P_3 = \infty$. There is one additional law: $\infty + P = P$ for all points P.

We add another point $\infty = (\infty, \infty)$ into this curve E, then the points of E form an **abelian group**. Abelian group is the key component of cryptography.

Here is the intuition of the addition equation above: Draw the line L through P_1 and P_2 (if $P_1 = P_2$, take the tangent line to E at P_1). The line L intesects E in a third point Q. Reflect Q through the x-axis (i.e., change y to $-y$) to get P_3. Figure 2 and Figure 3 illustrate the addition operation on an elliptic curve[6].

4 Elliptic Curve Discrete Logarithm Problem (ECDLP)

Suppose we have points P, P on an elliptic curve E and we know that $Q = kP$ $(= P + P + ... + P)$ for some integer k. When knows k and P, it is easy

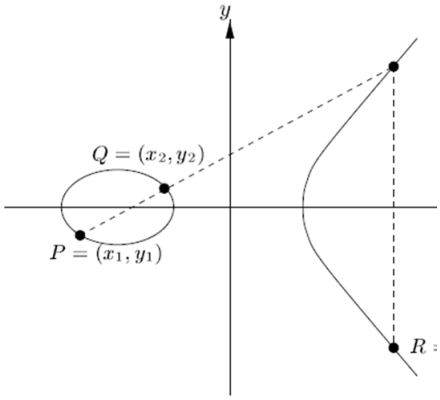

Figure 2: $P + Q = R$ $(P \neq Q)$

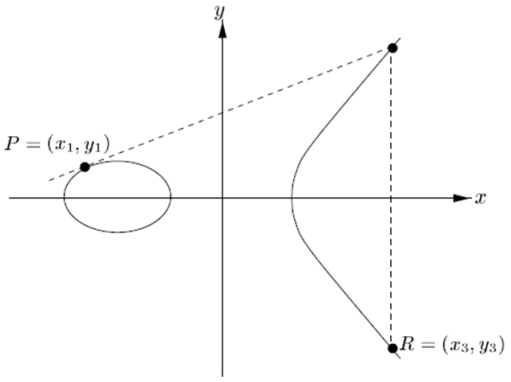

Figure 3: $P + P = 2P = R$

Alice	**Bob**
private key: k_a	private key: k_b
public key: k_aP	public key: k_bP
$k_aP \xrightarrow{to\ Bob}$	
	$\xleftarrow{to\ Alice} k_bP$
Shared key: $k_a * k_bP = k_ak_bP$	Shared key: $k_b * k_aP = k_ak_bP$

Figure 4: Model of Elliptic Curve Diffie Hellman

to compute Q, however, if only knows P and Q, it is hard to compute k. It is called the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**.

4.1 Applications

We can use ECDLP to share secrets between two parties while preventing any third party in the middle. For example, use Elliptic Curve Diffie Hellman (ECDH) to share a secret value k_ak_bP between Alice and Bob while preventing Eve in the middle. Suppose Alice has private key k_a and public key k_aP, Alice sends k_aP to Bob; Bob has private key k_b and public key k_bP, Bob sends k_bP to Alice. After that Alice can get the shared secret by compute $Shared\ secret = k_a * k_bP = k_ak_bP$; Bob can get the secret by compute $Shared\ secret = k_b * k_aP = k_ak_bP$. Eve in the middle can have k_aP and k_bP by sniffing the communication between Alice and Bob, but it is hard to get k_ak_bP from k_aP and k_bP. Figure 4 illustrates this model.

4.2 Bruteforce Attack

For any node P, it has a number n such that $nP = \infty$, therefore the choice of k is: $0 \leqslant k < n$. The bruteforce method is to enumerate all possible k until some

k such that $Q = kP$. The complexity of bruteforce attack is $O(n)$.

4.3 Pollard ρ Attack

Parallelized Pollard ρ attack is the best attacking method to general ECDLP (some attacks may have better performance towards some specific elliptic curve families)[1].

The basic idea is to find a pair of integers (a_1, b_1) and (a_2, b_2) such that

$$a_1P + b_1Q = a_2P + b_2Q$$

Then

$$(a_1 - a_2)P = (b_2 - b_1)Q$$

Hence

$$(a_1 - a_2) \equiv (b_2 - b_1)d \ (mod\ n)$$

Therefore

$$d = (a_1 - a_2)(b_2 - b_1)^{-1}\ mod\ n$$

n is the order of P ($nP = \infty$). Because we need the inverse of $(b_2 - b_1)^{-1}$, to guarantee any $(b_2 - b_1)$ has one and only one inverse, **n need to be a prime**.

To achieve this idea, we generate (a_0, b_0) randomly, let $X_0 = a_0P + b_0Q$ and

$$X_{i+1} = \begin{cases} X_i + P & X_i \in S_1 \\ 2X_i & X_i \in S_2 \\ X_i + Q & X_i \in S_3 \end{cases}$$

Record a_i and b_i of X_i, if any $X_i = X_{i'}$, then we can get $d = (a_i - a_{i'})(b_{i'} - b_i)^{-1}\ mod\ n$. The expected time complexity is $O(\sqrt{\pi n/2})$[7].

5 Implementation

5.1 Generate ECDLP parameters

The project is implemented in this way: first we implement high precision finite field operations; Then we randomly generate a prime p; generate $0 \leqslant a, b < p$; generate $0 \leqslant x \leq p$ that x has a corresponding y such that $P = (x, y) \in E$; Then compute the order of P: $order(P)$; generate $1 < k \leq order(P)$; compute $Q = kP$.

All parameters except k will be transferred to the attack component. The attack component computes and generates a number k_2 and returns. If $k_2 = k$, it means the attack is successful.

5.2 Attacks

First we implement the standalone bruteforce attack and standalone Pollard ρ attack.

After that we implement the parallelized bruteforce attack and Pollard ρ attack using Pthread.

We parallelize bruteforce attack as follows: Suppose the thread number is t, each thread will test around $thres = order(P)/t$ numbers. Each thread tests

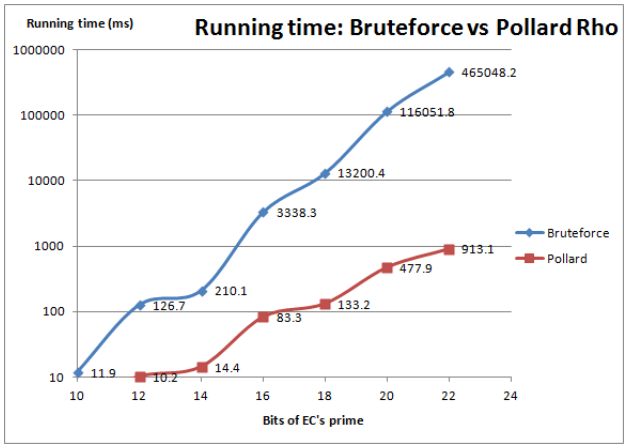

Figure 5: Running time of Bruteforce and Pollard under different bits of prime of Elliptic Curves

$min\ bound < k' \leqslant max\ bound$, if $Q = k'P$, then we found the k, then terminate all threads and return this k'.

For Pollard ρ method each thread has different random starting points $X_{0_i} = a_{0_i}P + b_{0_i}Q$. There is a global hash table stores all X, if any two X_{i_j} and $X_{i_{j'}}$ have the same value, we calculate corresponding k', terminate all threads and return.

6 Evaluation

We generate several datasets, for each dataset we generate 10 curves with the same bits of prime. For example, dataset $ecc12.txt$ has 10 curves, and each curve's prime is 12 bits. Generate 10 curves in a dataset because the average running time of 10 curves is the expected running time towards an elliptic curve of such size.

First we evaluate the speed of standalone bruteforce method and Pollard ρ method. We record the average running time under different bits of dataset of both bruteforce and Pollard ρ method, and draw them in Figure 5.

We can see that the running time is exponentially increasing when the bits of prime of the elliptic curve linearly increased. Also the speed of Pollard ρ method is hundreds of times faster than Bruteforce method.

Figure 6 shows the scaling up of parallelized Bruteforce method. The dataset is 20 bits primes of elliptic curves. We can see that it scales up sharply in first several times, but when thread number is near 8 the running time becomes almost the same.

Figure 7 shows the scaling up of parallelized Pollard ρ method. The dataset is 22 bits primes of elliptic curves. We can see that it scales up sharply in first several times, but when thread number is near 8 the running time becomes almost the same.

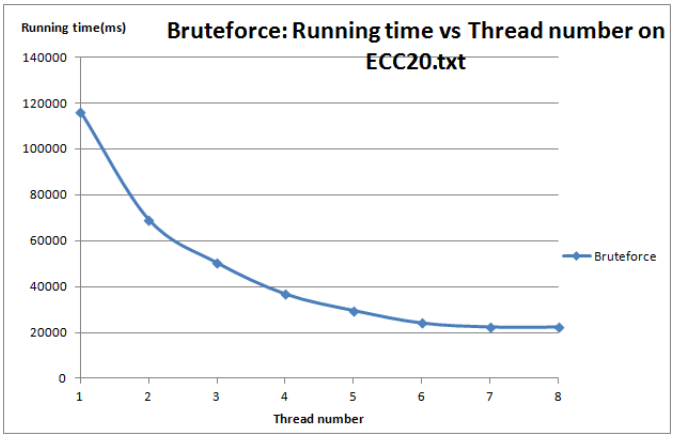

Figure 6: Running time of Bruteforce using different number of threads

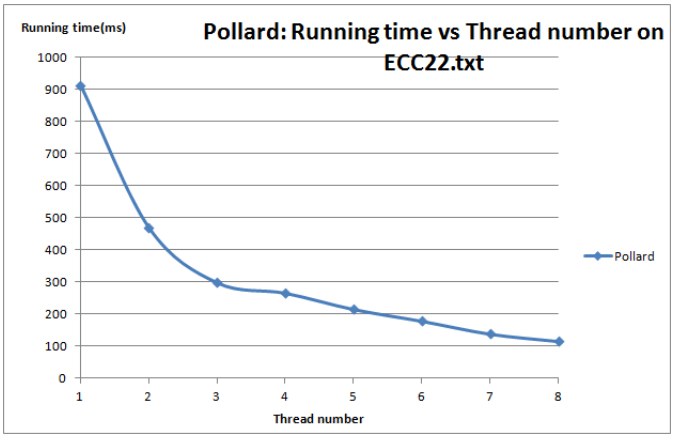

Figure 7: Running time of Pollard ρ using different number of threads

7 Future Work

Even though we did a complete work on parallelized attacks of ECDLP, we still have many improvements can be done:

1. The generator now is pretty stupid. To generate a good dataset we need $O(k * n)$ time, k is a constant (maybe hundreds, not trivially small) and n is the same size of prime. For t bits prime $n = 2^t$, which is big.

The reason why it is so large is because it is hard to get $order(P)$. There is a much faster method called Schoof's algorithm to get $order(P)$, but it is a complicate algorithm.

2. The random function can be improved. Actually in [1] we can use SHA-1 to form a better random function which achieves better randomness. Better randomness may improve the speed of Pollard ρ method.

3. We can parallelize it with more cores by using MPI and other more complicated mechanisms.

References

[1] "Certicom ECC Challenge", `http://www.certicom.com/images/pdfs/cert_ecc_challenge.pdf`

[2] Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions in Information Theory IT-22, pp. 644-654, 1976.

[3] D. Johnson, A. Menezes and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)", `http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf`

[4] ElGamal, T., "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Transactions on Information Theory, Vol. 30, No. 4, pp. 469-472, 1985.

[5] D. McGrew, K. Igoe and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC6090

[6] L. C. Washington, "Elliptic Curves Number Theory and Cryptography", Second Edition, 2008

[7] A. Escott, "Implementing a Parallel Pollard Rho Attack on ECC", `cacr.uwaterloo.ca/conferences/1998/ecc98/escott.ps`